Hadoop-Benchmark: Rapid Prototyping and Evaluation of Self-Adaptive Behaviors in Hadoop Clusters

Bo Zhang, Filip Krikava, Romain Rouvoy, Lionel Seinturier

Hadoop-Benchmark: Rapid Prototyping and Evaluation of Self-Adaptive Behaviors in Hadoop Clusters

Bo Zhang
University of Lille / INRIA
bo.zhang@inria.fr

Filip Křikava
Czech Technical University
filip.krikava@fit.cvut.cz

Romain Rouvoy
University of Lille / INRIA
romain.rouvoy@inria.fr

Lionel Seinturier
University of Lille / INRIA
lionel.seinturier@inria.fr

***Abstract*—Optimizing Hadoop executions has attracted a lot of research contributions in particular in the domain of self-adaptive software systems. However, these research efforts are often hindered by the complexity of Hadoop operation and the difficulty to reproduce experimental evaluations that makes it hard to compare different approaches to one another.**

To address this limitation, we propose a research acceleration platform for rapid prototyping and evaluation of self-adaptive behavior in Hadoop clusters. Essentially, it provides automated approach to provision reproducible Hadoop environments and execute acknowledged benchmarks. It is based on the state-of-the-art container technology that supports both distributed configurations as well as standalone single-host setups. We demonstrate the approach on a complete implementation of a concrete Hadoop self-adaptive case study.
The artifact is available at: https://github.com/Spirals-Team/hadoop-benchmark/raw/SEAMS17/artifact.zip

I. Introduction

One of the characteristics of many of Hadoop workloads is that their dynamics changes over time. Considering the scale of these workloads and the uncertainties of their execution environments, this can quickly lead to a waste of resources since the static configuration does not adapt to the current runtime condition. Optimizing Hadoop execution has therefore attracted a lot of research attention resulting in a number of different approaches in particular in the domain of self-adaptive software systems [1, 2, 4, 5, 6, 7, 8, 9]. However, this research effort is often hindered by the accidental complexity of setting representative Hadoop deployment in different distributed environments and comparing different approaches. Hadoop is a highly distributed system and correctly setting an operational cluster requires a significant amount of system administration knowledge and effort. Furthermore, there is currently no easy way to share and reproduce experimental evaluations of the existing self-adaptive approaches. It is therefore rather complicated to compare them since the experiments are hard to reproduce (*e.g.*, re-creating a testbed similar to the one used by the authors, availability of the implementations). This lack of repeatability and reproducibility can significantly affect scientific progress [10].

Majority of the self-adaptive approaches targeting Hadoop do not publicly share neither the configuration of their testbeds, nor the experimental results which in turn make them hard to reproduce and hard to repeat. The only option is then to contact the corresponding authors and try to repeat the experiment based on their suggestions and provided code which is usually tuned for a particular settings making the whole process difficult and error prone.

In this paper we address these limitations by proposing *hadoop-benchmark*[1], an open-source research acceleration platform for rapid prototyping and evaluation of self-adaptive behaviors in Hadoop clusters. The main objectives are to allow researchers to:

- *rapidly prototype—i.e.*, to experiment with self-adaptation in Hadoop clusters without the need to cope with low-level system infrastructure details,
- *reproduction—i.e.*, to share complete experiments for others to reproduce them independently, and
- *repetition—i.e.*, to experiment with and to compare their work, re-doing the same experiments on the same system using the same evaluation methods.

We achieve this by providing (1) a declarative mechanism to provision complete Hadoop clusters on either a local machine, a local cluster or in a number of cloud providers (*e.g.*, Google Cloud Engine, Microsoft Azure, Amazon AWS), (2) a systematic way how to package software artifacts (this can be the feedback control loop or just a set of configuration files), and (3) a number of pre-configured, well-known Hadoop benchmarks to easily assess the cluster performance.

The cluster provisioning and benchmark execution is done in an automated way based on simple configuration files which can be easily shared. The provisioned nodes in a cluster further includes monitoring service that can be used for developing touchpoints for system identification and the monitoring part of feedback control loops governing the self-adaptation.

To demonstrate the usage of the platform, we include a complete implementation of a Hadoop self-adaptation case study. Concretely, a feedback control loop that balances Hadoop job parallelism and throughput through Hadoop capacity scheduler adjustment based on our previous work [11].

It is important to note that while Hadoop has been mostly connected with the implementation of MapReduce paradigm it has grown and since version 2 it has become a general framework for distributed large-scale applications. Our focus

[1] https://github.com/Spirals-Team/hadoop-benchmark

on Hadoop goes therefore beyond MapReduce and has wide applications to other technologies that are based the core enabling technologies—*i.e.*, distributed files-systems (HDFS) and application scheduler (YARN). For example, the very same platform can be used to assess Apache Spark[2], Apache Tez[3] and other distributed data processing applications running on the top of a Hadoop cluster.

The rest of this paper is organized as follows. In Section II, we discuss the motivation behind the platform. Section III gives a brief overview of the key enabling technologies used for building the platform. Section IV presents the architecture and implementation. Section V illustrates the platform usage on a case study. Section VI discusses related work and provides a brief assessment of the proposed approach. Finally, Section VII concludes our work.

II. Motivation

There is a number of recurring tasks that a researcher must perform while experimenting with self-adaptation in distributed systems such as Hadoop. This includes: (1) *setting up a testbed—i.e.*, reserving a number of nodes in a local cluster or in a cloud, configuring the network between them and installing and configuring Hadoop distribution, (2) *setting up the self-adaptive behavior—i.e.*, getting all the touchpoints that monitors Hadoop and changes its configuration ready, (3) *running experiments—i.e.*, setting up and running a number of benchmarks, extracting data from the cluster, parsing the log files and conducting some data analysis over the measurements.

All of these tasks are both time consuming and require significant amount of domain-specific knowledge to operate all the software stacks involved. Due to the repetitive nature, researchers often develop numerous *adhoc* scripts to automate various tasks, but they are usually error prone. Keeping the experiment infrastructure in a desired state therefore requires a lot of manual effort.

Trying to compare one's result to some other self-adaptive approach makes things even harder. One has to get hold on the implementation and infrastructure configuration to be able to replicate the behavior. If the original experiment has been done in an cloud provider to which we do not have access, we are left to redo the configuration ourselves. Since many of the supporting tools are in form of *adhoc* scripts that are usually not portable (hard-coded paths, using commands available only in the original platform), reproduction becomes extremely complicated or even impossible. This is especially true if there is no support from the original authors of the experiment. There is therefore a need for a platform that will automate these tasks and simplify sharing of the experiments so that they can be reproduced by others. Concretely, the platform should support:

1) *Automated and rapid provisioning of complete Hadoop clusters.* Based on a single configuration that can be stored in a version control, it should be possible to provision a complete Hadoop environment in a number of cloud providers as well as on a local machine. The provisioning should be fast to reduce the time required for evaluating new approaches.
2) *Automated benchmark execution.* It should be possible to execute the common acknowledged Hadoop benchmarks (*i.e.*, hadoop-mapreduce-examples[4], HiBench[5], Swim[6]) to evaluate the performance of a given self-adaptive behavior.

Orthogonally to above features, the platform should be been designed with the focus on flexibility. It should allow one to experiment with the various aspects of Hadoop systems, provide hooks for the different life-cycle phases on the Hadoop components and the provide a possibility to develop required sensors and effectors.

III. Background

In this section we provide a brief overview of the core enabling technologies—*i.e.*, Hadoop and Docker.

A. Hadoop MapReduce Framework

Hadoop framework is composed of the two main subsystems: HDFS (*Hadoop Distributed File System*) and YARN (*Yet Another Resource Negotiator*). HDFS is an implementation of a distributed file-system that stores data across the nodes in a cluster. Typically, it uses one NameNode server that hosts the file-system index[7] and a number of DataNode servers that store the actual data blocks. YARN is a cluster-level computing resource manager responsible for resource allocations and jobs orchestration. It consists of one, per-cluster ResourceManager that acts as a global computing resource arbiter and a per-node NodeManager responsible for managing node-level resources.

Usual Hadoop cluster contains one controller node dedicated to the ResourceManager and NodeManager and a number of compute nodes for the workers running NodeManager and DataNode.

B. Docker Container Technology

Docker is an open-source project that aims at automating application deployment inside software containers. It achieves that through a concept of software containers, a layer of abstraction built on the top of system-level virtualization offered by Linux operating system. Essentially, a container offers a process-level resource isolation where each container has its own address space, file system and networking. Containers do not need any hypervisor (like in the case of classical virtualization), they are simply operating-system-level processes, yet completely separated from one another. A host machine can therefore run a number of them concurrently.

[2]https://spark.apache.org/

[3]https://tez.apache.org/

[4]https://github.com/apache/hadoop/tree/trunk/hadoop-mapreduce-project/hadoop-mapreduce-examples

[5]https://github.com/intel-hadoop/HiBench

[6]https://github.com/SWIMProjectUCB/SWIM

[7]often coupled with a secondary instance for high availability

A Docker container encapsulates some software package in a complete file-system that contains everything that software requires for it to run—*i.e.*, code, runtime, system tools, system libraries. A container is based on an image, a template that contains all the resources. They are created from base images using a simple domain-specific language describing a sequence of instructions (*e.g.*, execute a command, add a file to the image) that allows one to tailor the image to one's needs. For example, in our case, these steps include downloading Hadoop distribution, unpacking it and configuring it. From an image, a container is created by a Docker daemon. It instantiates the image, allocates a file-system and network interface, sets up an IP address and performs other tasks to bootstrap the container. The processes inside a container are fully isolated from the host machine like in the case of a classical virtual machine. However, it is just a process isolation and there is no hypervisor involved.

The main advantage of Docker versus classical virtualization is in the reproducibility and ability to deploy across a range of systems. Unlike a virtual machine image, a Docker image is composed of a set of configuration files that can all be stored in a version system. A built binary version of an image can be shared it via the DockerHub[8] infrastructure to save on building time. A Docker image is also technologically agnostic and can be run on any Docker host. We can therefore run the same container locally or on Amazon EC2. This is not the case with classical virtual machines technologies (*e.g.*, it is not possible to run Virtualbox image in Amazon EC2). Finally, spawning a container is order of magnitude faster than spawning a virtual machine (since there is no need to boot a new operating system) [3]. This greatly contributes to automation and reduces the number of steps that are needed to be performed manually. The hadoop-benchmark builds on the top of docker and further simplifies the orchestration of Docker containers for Hadoop clusters.

IV. Hadoop Benchmark Platform

The Hadoop Benchmark Platform is essentially a set of Docker images and scripts that orchestrate the provisioning and the execution of Docker hosts and containers deployed on these hosts. The design decision was to have a small code base with minimal dependencies. Besides the enabling technology—*i.e.*, docker and docker-machine—the platform therefore only requires bash and git.

There are three types of images: (1) *base images* that provide a vanilla Hadoop installation, (2) *extensions to the base images* with custom configuration coupled with implementation of some self-adaptive behavior, and (3) *benchmark images* that execute a particular benchmark suite. Currently, we provide one base image[9], one extension image implementing the case study (*cf.* Section V) and three benchmarks: hadoop-mapreduce-examples which include the implementation of the Hadoop canonical benchmarks such as wordcount or terasort, HiBench and Swim. All of these are well-known and accepted benchmarks in the community. Since the project is publicly available on GitHub, any contribution in forms of pull requests is welcomed. We hope that this will help to gradually extend the number of scenarios and benchmarks provided in order to foster the evaluation of self-adaptive approaches in Hadoop environments.

All the images are presented in two forms, a source form in the GitHub repository and a binary form in the DockerHub repository. The latter can be used to bypass manual image build and reuse the binary version that can be automatically downloaded by Docker daemon.

The base image contains a minimal Ubuntu operating system with Java and a vanilla Hadoop distribution. The only Hadoop settings we provide cover networking making sure all the Hadoop components can communicate with one another.

Before any of these images can be run, it is necessary to form a cluster of Docker hosts in which the images will run. Hadoop-benchmark facilitates it with a set of commands to manage the cluster life-cycle—*i.e.*, start, stop, restart, destroy. Based on a simple configuration (a number of hosts and details about their size and the environment where the nodes shall be created), it creates and configures all necessary Docker hosts. All the virtual machines will be also connected in a virtual network so Docker containers can communicate with one another.

For this task, we rely on *docker-machine*[10], a tool that creates virtual machines and install Docker engine on them. It currently supports creating virtual machines in local machine using either VirtualBox or VMWare, in local cluster, reusing existing machines simply via SSH, or in number of cloud providers[11] including all the major vendors. The main advantage of using Docker Machine is in the layer of abstraction it provides that can form the very same Hadoop cluster regardless the actual virtualization and cloud environment being used. Since the process is fully automated we can easily create multiple clusters allowing us to run experiments in parallel.

Figure 1 delivers a high-level architecture overview of a provisioned Hadoop cluster by the platform. Each cluster contains the following nodes (Docker hosts):

- One `hadoop-consul` which runs a single Docker container with the Consul service[12], a distributed key-value store used for service discovery. Concretely, it is being used as a backing mechanism for the virtual network formed along the provisioned hosts.
- One `hadoop-controller` which act as the controller for the Hadoop cluster. It runs two containers: `controller` and `graphite`. The former provides the ResourceManager and NameNode services. The latter contains the Graphite service[13] for real-time visualization

[8]https://hub.docker.com/

[9]which is split in two to shorten the download and build times

[10]https://docs.docker.com/machine

[11]A list of supported providers https://docs.docker.com/machine/drivers

[12]https://www.consul.io

[13]http://graphite.wikidot.com

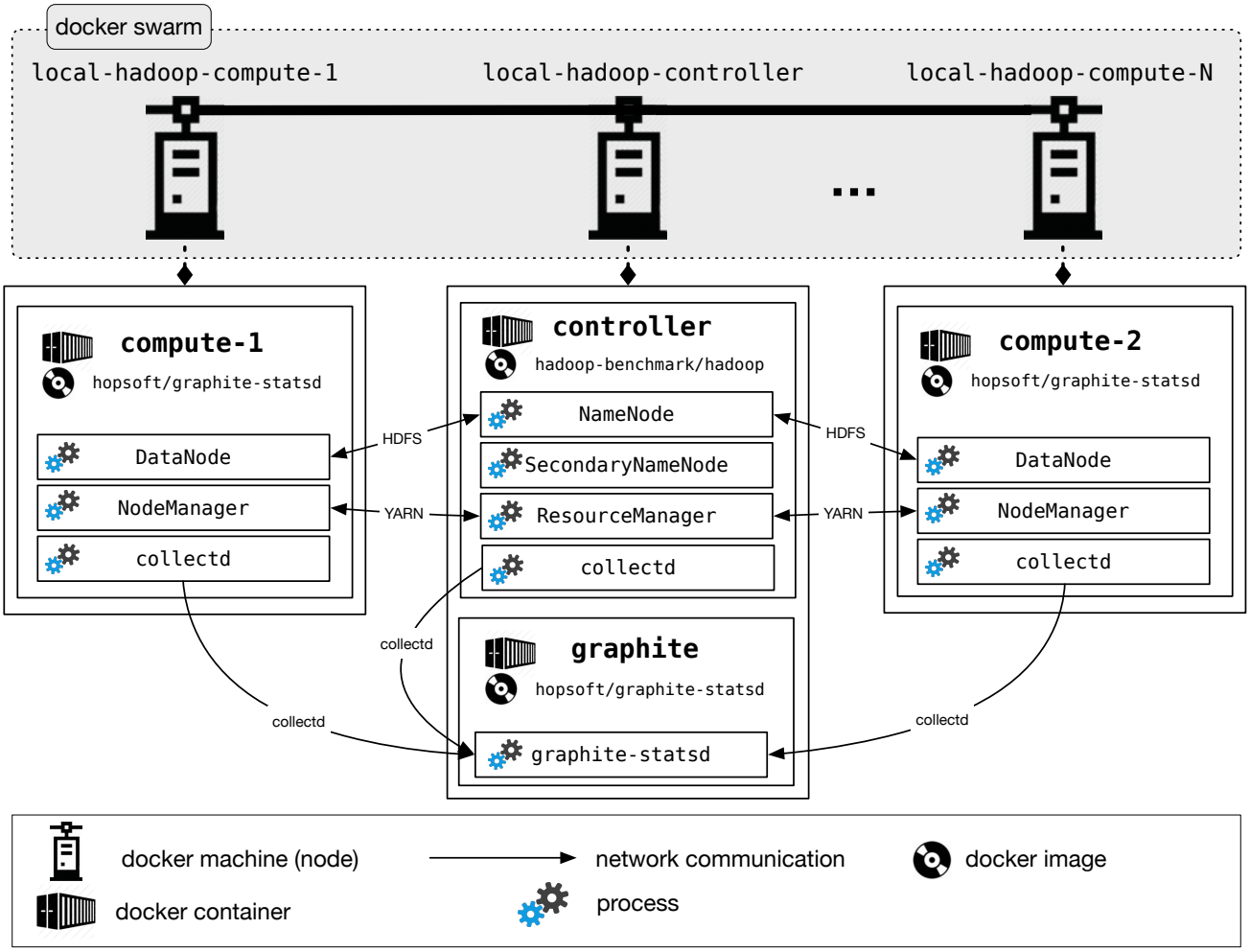

Fig. 1. High-level overview of the provisioned cluster using hadoop-benchmark

of monitoring data coming from other containers (*e.g.*, CPU, memory, I/O).

– One or mode `hadoop-compute` nodes which represents the Hadoop working nodes. They run a single container with NodeManager and DataNode services.

The `hadoop-contoller` and `hadoop-compute` nodes are connected into *Docker Swarm*, a native clustering mechanism for Docker. Among others, this allows them to be part of the same network and communicate together. The Docker Swarm uses the distributed key-value store, Consul in our case, to keep track of which nodes are available in the cluster.

Each node in the cluster is also equipped with a monitoring service, *collectd*[14]. It collects a common set of performance-related metrics and make them available in CSV and RRD[15] format. They can be easily fed into any monitoring part of a feedback control loop. The `graphite` service provides a web-based user interface of observe these metrics at runtime.

A performance of such provisioned Hadoop cluster can be assessed by a number of benchmarks. Each is implemented as a docker image and runs in a standalone container alongside the `controller`. The platform also allows to quickly access logs of any of the services from the client machine as well as to access a shell inside any of the running container. Data from the containers are available to be mounted on a client machine or to any additional containers created within the same network.

In addition, Hadoop-Benchmark also provides some R scripts for the subsequent analysis of the results generated by the packaged benchmarks. The required R packages are therefore optional for the users.

V. Case Study

As a use case to show the capabilities of the platform, we choose a self-adaptive behavior that automatically balances the job parallelism and throughput in a Hadoop cluster [11]. Concretely, it adjusts a YARN capacity scheduler parameter, MARP[16], which controls the ratio between the number of

[14]https://collectd.org

[15]https://en.wikipedia.org/wiki/RRDtool

[16]Maximum Application Master Resource in Percent

concurrently executing MapReduce jobs versus the number of running map and reduce tasks. This is one of the crucial parameter whose inappropriate configuration can have a serious impact on Hadoop performance (up to 40%, *cf.* Zhang *et al.* [11]).

The proposed platform accelerates the implementation in the following ways:

(1) First, it helps us to validate the hypothesis—*i.e.*, the impact of MARP on the cluster performance. The platform quickly provisions a Hadoop cluster and allows us to run a series of experiments using the standard Hadoop benchmarks. For each experiment, we change the MARP value and observe its effect. The results are shown in Figure 2.

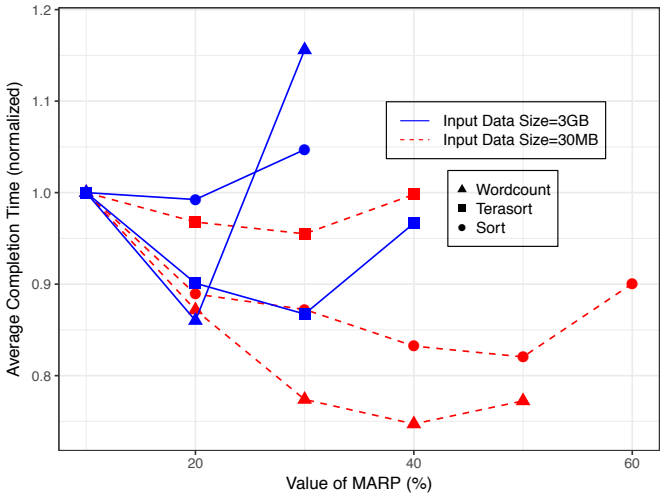

Fig. 2. Effects of different MARP configurations, job type and job size on mean completion time of 100 jobs.

(2) Once the hypothesis is validated, we can prototype the actual implementation. To make this easily reproducible, we create a new docker image that extends from the platform base image and include the implementation code together with a start-up hook. Concretely, we create a feedback control loop in Java that periodically observes the cluster memory usage and based on its value, it proportionally adjusts the MARP value. It uses a proportional controller coupled with a Kalman filter to control the MARP value. The start-up hook is a shell script that launches the Java application after the ResourceManager is started. The cluster memory usage is extracted from the ResourceManager log file to which all NodeManagers report periodically their memory statistics.

(3) To tune the controller, we rerun the same benchmarks while varying the controller settings. We can leverage from the ability to execute experiments in multiple clusters in parallel to save time. For example the SWIM benchmark can run for hours.

(4) The performance gain can be compared by simply repeating the same experiments twice. Once on the vanilla Hadoop and once using our controller. A sample result from Hadoop cluster made of 11 physical hosts[17] (1 control node and 10 compute nodes) deployed on the GRID5000 infrastructure [18] is shown in Figure 3.

(5) Having used the hadoop-benchmark, we can easily share the code as well as the configuration on GitHub[19] so other researchers who are interested in this approach can get it and repeat it.

The docker image of this self-balancing scenario extends from the base images. We install our self-balancing approach into the base images while modifying the entrypoint of the new docker image of self-balancing scenario. Once Hadoop cluster is provisioned, Hadoop-Benchmark can therefore launch our approach to autonomously optimize Hadoop performance.

To create own custom docker images, users only need to follow the 2 steps: 1) copy their compiled approachs into the base images, and then 2) update the entrypoint of new docker images to automatically start the approachs. After the new custom images are created, users can re-create their custom Hadoop cluster by using Hadoop-Benchmark with custom terminal environment. This step is similar to the tutorial creating self-balancing scenario.

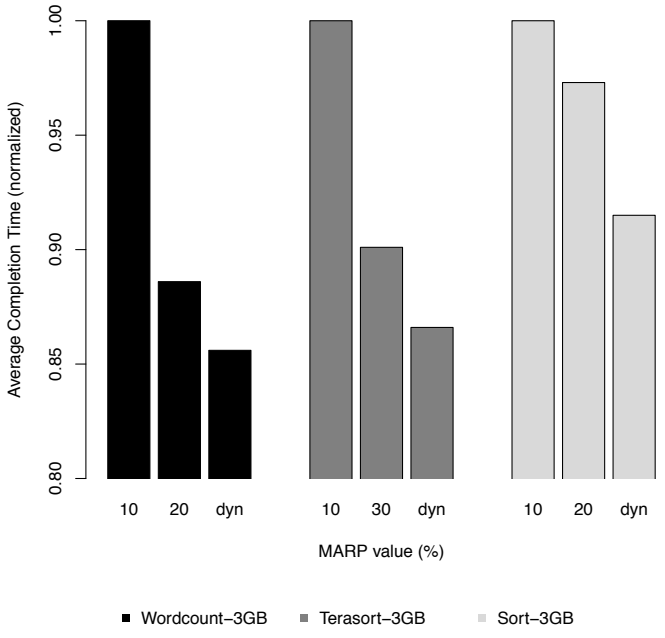

Fig. 3. Performance comparisons of 3 HIBENCH. The first bar corresponds to the vanilla configuration—*i.e.*, 10%—the second to the best statistically profiled value, and the last to our self-balancing approach.

VI. DISCUSSION

In this section we overview some of the existing works and provide a brief assessment of our approach.

[17] 2 Intel Xeon L5420 CPUs, 4 cores, 15GB RAM, 298GB HDD

[18] http://grid5000.fr

[19] https://github.com/Spirals-Team/hadoop-benchmark/tree/master/scenarios/self-balancing-example

A. Related Work

Hadoop is a well-establish data processing framework that is being used extensively in both academia and industry. As such, there exists many commercial and open-source approaches for its provisioning. Nearly each cloud provider provides some support for assembling a Hadoop cluster within its infrastructure ranging from detailed step-by-step tutorial to fully-featured web applications. The major development operation tools, such as Ansible[20], Puppet[21] or Chef[22], also contain a set of provisioning scripts for creating Hadoop deployment. Finally, Cloudera Manager[23] and Ambari[24] are the two most popular tools dedicated for provisioning, managing, and monitoring Hadoop clusters.

Despite that there exists a number of tools that automate Hadoop deployment, they all focus on provisioning long-term production-ready operating clusters. Instead, our approach aims at deployment of experimental environments that are used to rapidly evaluate different Hadoop configurations and adaptation approaches. Furthermore, the existing approaches are usual tight to a particular Hadoop version or configuration, or they bring a large stack of other dependencies. While this is useful for production-ready clusters, it slows the deployment and complicates experimentation. Finally, a reproduction of an experiment from a cluster provisioned by one of these tools in another environment is again hindered. It requires an access to the particular tool and some level of knowledge how to reconfigure it to fit the needs of the target environment.

B. Assessment

This work has been driven by the need of a platform that can be used to rapidly prototype self-adaptation in Hadoop clusters and allows one to easily share resulting experiments. The two main objectives were to provide (1) an automated way to rapidly provision Hadoop clusters, and (2) an automated way to execute Hadoop benchmarks. Both with a focus on reproducibility and repeatability.

The automated provisioning is facilitated by the orchestration of docker-machine and Docker containers. This allows us to deploy Hadoop transparently in a number of environments ranging from local machine, local existing cluster to major cloud providers. All deployments are based on the same simple configuration. While there is some overhead induced by the creating virtual machines in the case of local deployment or deployment in cloud, this only needs to happen once. The actual provisioning of Hadoop is based on docker containers and spawning a container is close to spawning a regular operating system process—*i.e.*, the overhead is minimal. This allows us to provision a complete new Hadoop cluster in a matter of seconds. This is considerably faster than the other solutions that require full redeployment of virtual machine (in order to make sure there are no stalled data from previous deployment) which may take from dozens of minutes to hours.

The automated benchmark execution is similarly based on Docker containers. Currently, we provide three well-known Hadoop benchmarks: `hadoop-mapreduce-examples`, HiBench and Swim.

All steps from assembling the cluster to executing a benchmark is driven by configuration files and few bash scripts with no other dependencies. Any changes can therefore be kept in plain text files that are be stored in version control. The project is open-source and the complete source code is hosted on GitHub: https://github.com/Spirals-Team/hadoop-benchmark.

VII. Conclusion

This paper presents an implementation of an open and reproducible testbed for the prototyping and the evaluation of self-adaptive behaviors in Hadoop clusters. While Hadoop is acknowledged as a *de facto* standard for the processing of large-scale dataset, the performances of Hadoop clusters tend to be impacted by the underlying infrastructure as well as the considered workloads and algorithms. Optimizing Hadoop executions has therefore attracted a lot of research attention, in particular in the domain of self-adaptive software systems [1, 2, 4, 6, 8, 9]. Yet, reproducing and assessing the proposed contributions might quickly be hindered by the accidental complexity of Hadoop deployments.

Our solution leverages the state of practice in lightweight virtualisation techniques to deliver a flexible approach to advance the research in the software engineering of self-adaptive Hadoop systems. The resulting platform allows researchers to experiment with self-adaptive behavior in Hadoop clusters and create repeatable and reproducible experiments. We demonstrate the usability of our approach on an implementation of a concrete case study that autonomously adjusts the YARN capacity scheduler to appropriately balance the MapReduce jobs throughput and parallelism. Beyond this demonstration, we believe that this research asset can benefit the research community by providing a common environment to empirically compare the research contributions in the area of self-adaptive applications in Hadoop clusters. The design and implementation choices we made leverage the reuse and the extension of this environment in order to fit a large diversity of scenarios.

For the future work, we plan to add more scenarios with various sensors and actuators to enrich the Hadoop-based environments of Hadoop-Benchmark. Moreover, many other distributed systems based on YARN (*e.g.*, Spark) can also be packaged into this work. Hadoop-Benchmark can therefore benefit more research of system-level self-adaptation in different environments.

Acknowledgments. This work has been supported by the Datalyse project: www.datalyse.fr and by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation program (grant agreement No 695412). Experiments presented in this paper were carried

[20]http://ansible.com
[21]http://puppetlabs.com
[22]http://chef.io
[23]http://cloudera.com/products/cloudera-manager.html
[24]https://ambari.apache.org

out using the Grid'5000 testbed, supported by a scientific interest group hosted by Inria and including CNRS, RENATER and several Universities as well as other organizations (see https://www.grid5000.fr)

REFERENCES

[1] M. Berekmeri, D. Serrano, S. Bouchenak, N. Marchand, and B. Robu. A control approach for performance of big data systems. In *IFAC World Congress*, volume 19, 2014.

[2] K. Chen, J. Powers, S. Guo, and F. Tian. CRESP: Towards Optimal Resource Provisioning for MapReduce Computing in Public Clouds. *IEEE Trans. Parallel Distrib. Syst.*, 2014.

[3] W. Felter, A. Ferreira, R. Rajamony, and J. Rubio. An updated performance comparison of virtual machines and linux containers. *IBM technology*, 28:32.

[4] B. Ghit, N. Yigitbasi, A. Iosup, and D. H. J. Epema. Balanced resource allocations across multiple dynamic MapReduce clusters. In *ACM SIGMETRICS*, 2014.

[5] H. Herodotou, H. Lim, G. Luo, and N. Borisov. Starfish: A Self-tuning System for Big Data Analytics. *Conference on Innovative Data Systems Research*, 2011.

[6] P. Lama and X. Zhou. AROMA: automated resource allocation and configuration of mapreduce environment in the cloud. In *ICAC*, 2012.

[7] G. Liao, K. Datta, and T. L. Willke. Gunther: Search-Based Auto-tuning of MapReduce. In *Euro-Par 2013 Parallel Processing*, 2013.

[8] P. Padala, K. Hou, K. G. Shin, X. Zhu, M. Uysal, Z. Wang, S. Singhal, and A. Merchant. Automated control of multiple virtualized resources. In *Proceedings of the 2009 EuroSys*.

[9] P. Padala, K. G. Shin, X. Zhu, M. Uysal, Z. Wang, S. Singhal, A. Merchant, and K. Salem. Adaptive control of virtualized resources in utility computing environments. In *Proceedings of the 2007 EuroSys*.

[10] J. Vitek and T. Kalibera. R3: Repeatability, reproducibility and rigor. *SIGPLAN Not.*, 47(4a):30–36, Mar. 2012.

[11] B. Zhang, F. Křikava, R. Rouvoy, and L. Seinturier. Self-Balancing Job Parallelism and Throughput in Hadoop. In *16th International Conference on Distributed Applications and Interoperable Systems (DAIS)*, volume LNCS-9687, pages 129–143, Heraklion, Crete, Greece, June 2016. Springer.